SISTEMI OPERATIVI

ESERCIZIO N. 1 dell'8 NOVEMBRE 2002

In un monastero, i **frati amanuensi** gestiscono **L libri**. I frati si dedicano alla copiatura dei libri: richiedono al **bibliotecario** l'accesso ai libri, li usano e poi li restituiscono. Gli amanuensi usano 2 libri per volta, leggendo dal primo e scrivendo sul secondo. In generale, il bibliotecario consente la lettura concorrente di un libro da parte di più amanuensi, mentre permette che solo un amanuense alla volta scriva su un libro, ed inoltre non è possibile leggere e scrivere sullo stesso libro contemporaneamente. Gli amanuensi sono di due tipi: **superiori** o **semplici**. I primi hanno priorità sui secondi.
Si implementi una soluzione usando il costrutto monitor per modellare il **bibliotecario** e i processi per modellare gli **amanuensi** e si descriva la sincronizzazione tra i processi. Nella soluzione si massimizzi l'utilizzo delle risorse. Si discuta se la soluzione proposta può presentare starvation e in caso positivo per quali processi, e si propongano modifiche e/o aggiunte per evitare starvation.

```
program Monastero

const     L = ...; { numero di libri }
type      libro = 1..L;
type      tipo = (semplice, superiore);

type amanuense = process (t: tipo; l1, l2: libro)
begin
      repeat
            s.richiedi (t, l1, l2);
            < legge da l1 e scrive su l2 >
            s.rilascia (l1, l2);
      until false
end

type bibliotecario = monitor

{ variabili del monitor }
var  sospesi: array[tipo] of integer;
      { numero di amanuensi sospesi }
      coda : array[tipo] of condition;
      { code su cui sospendere gli amanuensi }
      lettori : array[libro] of integer;
      { numero di lettori per libro }
      scrittore : array[libro] of boolean;
      { dice se i libri sono acceduti in scrittura }

procedure entry richiedi (t: tipo; l1, l2: libro)
begin
      while (scrittore[l1] or { se c'è uno scrittore su l1}
        lettori[l2] > 0 or { o un lettore su l2}
        scrittore[l2] > 0 or { o uno scrittore su l2}
        (t = semplice and coda[superiore].queue) do
        { o c'è un superiore in coda }
      begin
            sospesi[t] ++;
```

```
            coda[t].wait;
            sospesi[t] --;
        end

        { acquisisce la risorsa }
        lettori[l1] ++;
        scrittore[l2] := true ;
end

procedure entry rilascia (l1, l2: libro)
var s, i: integer;
begin
        { rilascia la risorsa }
        lettori[l1] --;
        scrittore[l2] := false ;

        { risveglia prima gli amanuensi superiori }
        s := sospesi[superiore];
        for i := 1 to s do
            coda[superiore].signal;
        { poi quelli semplici }
        s := sospesi[semplice];
        for i := 1 to s do
            coda[semplice].signal;
end

begin { inizializzazione delle variabili }
        sospesi[superiore] := 0;
        sospesi[semplice] := 0;
        for i := 1 to L do
        begin
            lettori[i] := 0;
            scrittore[i] := false;
        end
end
```

```
var  s: bibliotecario; { il nostro monitor }
     su1, su2, ... : amanuense (superiore, k, l);
     se1, se2, ... : amanuense (semplice, j, n);

begin end.
```

Starvation

La soluzione proposta presenta starvation nei confronti degli amanuensi semplici, i quali possono essere scavalcati in modo indefinito dagli amanuensi superiori.
Per evitare ciò, si può imporre di alternare la priorità ogni tot di esecuzioni, tenendone conto tramite un contatore.

NOTE

Poiché gli amanuensi hanno bisogno di due libri, si è preferito sospenderli tutti in una unica coda, risvegliarli tutti quando un libro si libera, e lasciare che siano essi stessi a ritestare le condizioni in un ciclo while.

2° soluzione che sfrutta meglio le risorse

```
program Monastero

const     L = ...; { numero di libri }
type      libro = 1..L;
type      tipo = (semplice, superiore);

type amanuense = process (t: tipo; l1, l2: libro)
{ come prima }

type bibliotecario = monitor

{ variabili del monitor: come prima più }
var  LetSupAtt : array[libro] of integer;
     { superiori in attesa di leggere un libro }
     ScrSupAtt: array[libro] of integer;
     { superiori in attesa di scrivere un libro }

procedure entry richiedi (t: tipo; l1, l2: libro)
begin
     if (t = semplice)
     begin
          while (scrittore[l1] or { se c'è uno scrittore su l1}
           lettori[l2] > 0 or { o un lettore su l2}
           scrittore[l2] > 0 or { o uno scrittore su l2}
           ScrSupAtt[l1] > 0 or ScrSupAtt[l2]
           or LetSupAtt[l2] do
           { o c'è un superiore che aspetta i libri richiesti }
          begin
               sospesi[t] ++;
               coda[t].wait;
               sospesi[t] --;
          end
     end
```

```
        else { t = superiore }
        begin
            while (scrittore[l1] or { se c'è uno scrittore su l1}
             lettori[l2] > 0 or { o un lettore su l2}
             scrittore[l2] > 0 do { o uno scrittore su l2}
            begin
                sospesi[t] ++;
                LetSupAtt[l1] ++;
                ScrSupAtt[l2] ++;
                coda[t].wait;
                LetSupAtt[l1] --;
                ScrSupAtt[l2] --;
                sospesi[t] --;
            end
        end

        { acquisisce la risorsa }
        lettori[l1] ++;
        scrittore[l2] := true ;
end

procedure entry rilascia (l1, l2: libro) { come prima }

begin { inizializzazione delle variabili come prima più }
        for i := 1 to L do
        begin LetSupAtt[i] := 0; ScrSupAtt[i] := 0; end
end

var  { come prima }
begin end.
```